# CS 580 Client-Server Programming
Fall Semester, 2012
Doc 24 Client-Server Shortcuts
Dec 4, 2012

# Client-Server Shortcuts

Remote procedure calls

Distributed objects

Web & Http

Frameworks

# Remote procedure calls

Client "directly" calls a function on the server

Issues

Cross platform

Marshalling/unmarshalling of parameters and results

How can one handle pointers as parameters?

Different contexts of client and server

Registering and finding servers

# Example - Add Server

```
import org.apache.xmlrpc.*;

public class AddServer {
       public Integer addtwo(int x, int y)  {
        return new Integer( x + y);
     }

    public static void main( String[] args)  {
        try {
            System.out.println("Starting server on port 8080");
            WebServer addTwoServer = new WebServer(8080);
            addTwoServer.addHandler("examples", new AddServer());
            addTwoServer.start();
            System.out.println("server running");
        }
         catch (Exception webServerStartError) {
            System.err.println( "JavaServer " + webServerStartError.toString());
          }
    }
}
```

Client can access all public instance methods in AddServer

# Example - Client

```
import java.util.*;
import org.apache.xmlrpc.*;

public class XmlRpcExample {
        public static void main (String args[])  {
         try {
            XmlRpcClient xmlrpc =  new XmlRpcClientLite("http://127.0.0.1:8080/");
                    Vector parameters = new Vector ();
            parameters.addElement (new Integer(5) );
            parameters.addElement (new Integer(3) );

            Integer sum = (Integer) xmlrpc.execute("examples.addtwo", parameters);

            System.out.println( sum.intValue() );
           } catch (java.net.MalformedURLException badAddress) {
            badAddress.printStackTrace( System.out);
           } catch (java.io.IOException connectionProblem) {
            connectionProblem.printStackTrace( System.out);
           } catch (Exception serverProblem) {
            serverProblem.printStackTrace( System.out);
           }
        }
}
```

# Note

No explicit sockets
No parsing
Worker pool done for us

Protocol design - just public methods of Server object

# Client program has to know

Server machine name or IP
Path to server program
Name of remote method
Number, Type and Order of arguments

# Consequences

## Benefits

Protocol = public methods
Handles the network communications
Handles generation/parsing of messages
Multiple language support
Platform independent
Simple

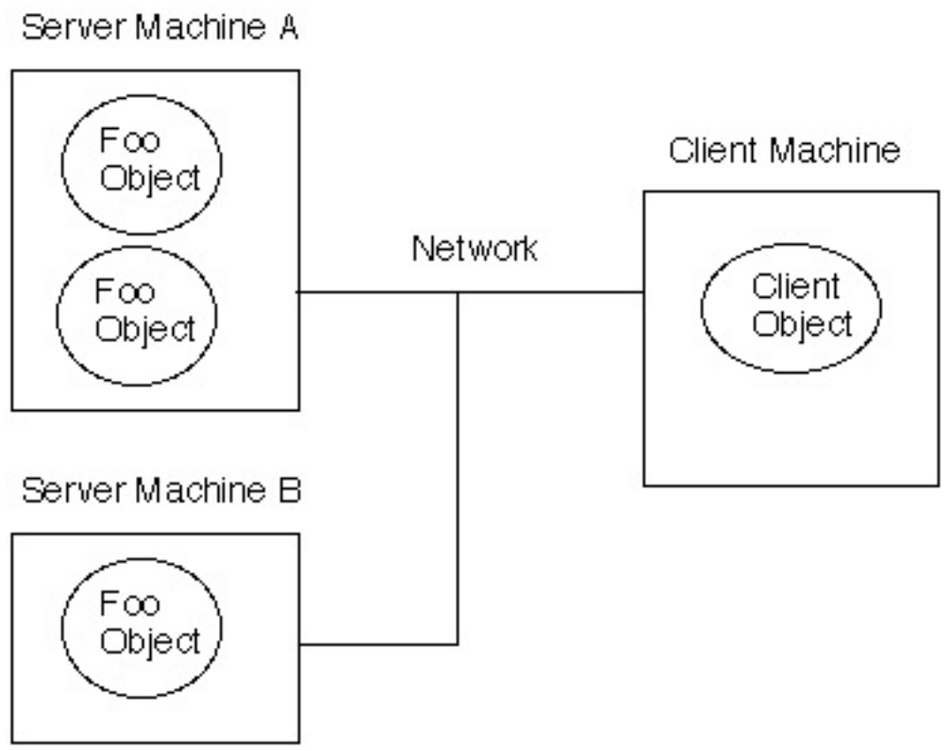
## Drawbacks

Long messages
Limited support for objects

# Distributed Objects

# Distributed Objects

System that allows sending of messags to objects on remote machines




```
public class Foo {
   public String hello() { return "Hi there"; }
}
```

```
Foo remote = getRemoteObject();
String message = remote.hello();
```

# Some Existing Systems

Java RMI
CORBA
DCOM
Pyro (Python)
dRuby
ReplicaNet (C++)

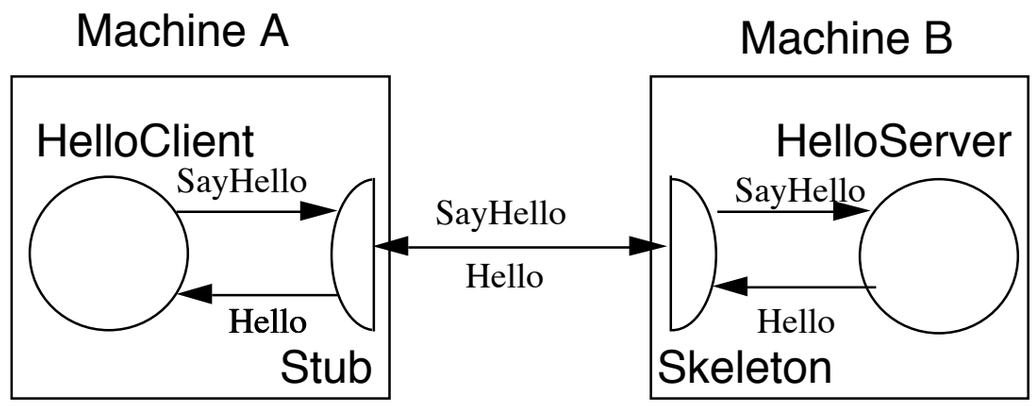
# RMI - Hello World Example

Implement a server with the method sayHello()

Parts needed

- Hello interface
- Client code
- Server Code
- rmiregistery
- Proxy classes

(Permission file)



# The Remote Interface

```
public interface Hello extends java.rmi.Remote
    {
    String sayHello() throws java.rmi.RemoteException;
    }
```

# HelloServer

```
import java.net.InetAddress;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloServer extends UnicastRemoteObject implements Hello {

    public HelloServer() throws RemoteException {  }

    public String sayHello() { return  "Hello World from " + getHostName(); }

    protected static String getHostName() {
        try {
            return InetAddress.getLocalHost().getHostName();
        }
        catch (java.net.UnknownHostException who)  {
            return "Unknown";
        }
    }
}
```





Example modified from Getting Started Using Java RMI, http://java.sun.com/javase/6/docs/technotes/guides/rmi/hello/hello-world.html

# HelloServer Continued

```
    public static void main(String args[]) {
        try {
            Server helloServer = new Server();
            Hello stub = (Hello)
UnicastRemoteObject.exportObject(helloServer, 0);

            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", stub);

            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
```

# HelloClient

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

    public static void main(String[] args) {

        String host = (args.length < 1) ? "localhost" : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```





Example from Getting Started Using Java RMI, http://java.sun.com/javase/6/docs/technotes/guides/rmi/hello/hello-world.html

# Web Services

SOAP – Simple Object Access Protocol

1998 Created by Winer, Box, Atkinson, Al-Ghosein

Version 1.2 dropped the acronym

WSDL – Web Services Description Language

UUDI – Universal Description, Discovery and Integration of Web Services

# Sample Server

```
package samples.quickstart.service.pojo;

import java.util.HashMap;

public class StockQuoteService {
    private HashMap map = new HashMap();

    public double getPrice(String symbol) {
        Double price = (Double) map.get(symbol);
        if(price != null){
            return price.doubleValue();
        }
        return 42.00;
    }

    public void update(String symbol, double price) {
        map.put(symbol, new Double(price));
    }
}
```





Standard example from Apache Axis SOAP server

# Some Performance

Time in seconds

| | Connect time | Send String 21,000 Chars | Send 5,000 integers | Server LOC | Message size sending 100 integers |
|---|---|---|---|---|---|
| socket | 0.002242 | 0.001377 | 6.71 | 25 | 85,863 |
| Corba | 0.000734 | 0.004601 | 1.52 | 18 | 27,181 |
| XML-RPC | 0.007040 | 0.082755 | 100.34 | 17 | 324,989 |
| SOAP | 0.000610 | 0.294198 | 1,324.30 | 10 | 380,288 |

Factor slower/larger than using Socket

| | Connect time | Send String 21,000 Chars | Send 5,000 integers | Server LOC | Message size sending 100 integers |
|---|---|---|---|---|---|
| Corba | 0.3 | 3.3 | 0.2 | 0.7 | 0.3 |
| XML-RPC | 3.1 | 60.1 | 15.0 | 0.7 | 3.8 |
| SOAP | 0.3 | 213.7 | 197.4 | 0.4 | 4.4 |

Code written in Python

http://www-128.ibm.com/developerworks/webservices/library/ws-pyth9/

# REST

http://developers.slashdot.org/article.pl?sid=03/04/03/1942235&mode=nocomment&tid=185&tid=156

tadghin:

“I was recently talking with Jeff Barr, creator of syndic8 and now Amazon's chief web services evangelist.  He let drop an interesting tidbit.  Amazon has both SOAP and REST interfaces to their web services, and 85% of their usage is of the REST interface."

" Despite all of the corporate hype over the SOAP stack, this is pretty compelling evidence that developers like the simpler REST approach. “

# History

Roy Fielding

2000 Ph.D. Thesis

Architectural Styles and the Design of Network-based Software Architectures

What makes the Web scale?

# REST Principles

Application state and functionality are abstracted into resources

Resource is uniquely addressable using a link

All resources share a uniform interface for the transfer of state between client and resource, consisting of

- A constrained set of well-defined operations
- A constrained set of content types, optionally supporting code on demand

A protocol which is:

- Client-server
- Stateless
- Cacheable
- Layered

# Common Usage

Server

- Web server
- Returns data in JSON format

Client

- Makes http(s) request
- Uses library to read JSON data

# Parse

http://parse.com

Backend for Web, Mac, iOS & Android Apps

NoSQL database

Handles network connection

You don't write any backend code

# Client Example - Creating and saving

```
ParseObject newTeacher = new ParseObject("Instructor");
newTeacher.put("firstName", "Peter");
newTeacher.put("lastName", "Gun");
newTeacher.put("email", "bullet@gun.com");
newTeacher.put("office", "GMCS 723");
newTeacher.put("phone", "619-594-0000");
newTeacher.save();

ParseObject comment = new ParseObject("Comment");
comment.put("text", "Sample Comment");
comment.put("date", new Date().toString());
comment.put("parent", newTeacher);
comment.save();
```

No backend code written for this app





The two object newTeacher and comment are now saved on Parse server and can be accessed by other devices

# Fetching Objects

```
ParseQuery getInstructor = new ParseQuery("Instructor");
    getInstructor.whereEqualTo("lastName", "Gun");
    ParseObject gun = getInstructor.getFirst();

    ParseQuery comments = new ParseQuery("Comment");
    comments.whereEqualTo("parent", gun);

    List<ParseObject> commentList = comments.find();
    for (ParseObject comment : commentList) {
          do something with each comment
    }
```

# Parse Apps, IDs & Client Keys

You create a Parse App to store data for each app

You get an app ID and client key for each Parse App

Android app

- Needs the app ID and client key to access the data for that app
- App can only access data in that app

```
Parse.initialize(this, "AppId", "ClientKey");
```

# "Tables"

If does not exist creates Instructor "Table" on parse server

```
ParseObject newTeacher = new ParseObject("Instructor");
newTeacher.put("firstName", "Peter");
newTeacher.put("lastName", "Gun");
newTeacher.put("email", "bullet@gun.com");
newTeacher.put("office", "GMCS 723");
newTeacher.put("phone", "619-594-0000");
newTeacher.save();
```

## Parse web data view

Classes

+ Row | - Row | + Col | - Col | More ▾ | 1 - 20 of 29 rows

- Comment 2915
- GameScore 3
- Installation 0
- Instructor 29
- User 1

| objectId | email string | firstName string | id number | instructorId num... | lastName string |
|---|---|---|---|---|---|
| kw7RVTAtTX | bullet@gun.com | Peter | kw7RVTAtTX | (empty) | Gun |
| GPsWiF02Su | G.E.Whittenburg... | Gene | GPsWiF02Su | 28 | Whittenburg |
| CLSXcWV... | carol.venable@sd... | Carol | CLSXcWVcDT | 27 | Venable |
| suHjspB4VY | Snyder1@mail.sd... | Will | suHjspB4VY | 26 | Snyder |
| XDB3AwKidQ | psager@mail.sds... | Paul | XDB3AwKidQ | 25 | Sager |
| BE0Ihv9NpF | drno@mail.sdsu.edu | Nathan | BE0Ihv9NpF | 24 | Oestreich |
| LE018d7gjA | sharon.lightner@s... | Sharon | LE018d7gjA | 23 | Lightner |

# Rate Your Instructor Example

Moved all the data from server for assignment 2&3 to Parse objects

Issues

- No server side logic
- Ratings was computed on server

Milliseconds to read 450 comments from server

| My Server | Parse Serer |
|---|---|
| 464 | 848 |
| 389 | 815 |
| 410 | 934 |
| 440 | 678 |

Milliseconds to write 10 comments to server

| My Server | Parse Serer |
|---|---|
| 610 | 7469 |
| 479 | 6811 |
| 412 | 6813 |
| 475 | 6789 |

# Frameworks

# Node.js

http://nodejs.org/

JavaScript on desktop/server side

Event-driven non-blocking I/O framework for servers

Scalable network programs

Uses Googles V8 JavaScript Engine

Compiles JavaScript to machine code

Used in HP's WebOS Phones and tablets

# Threads

Common way to scale performance

While one thread is blocked on I/O another thread can perform work

Typical server

- One high priority thread accepts connects from clients
- Once accepted a client connection is give to a worker thread

# Thread Issues

- Overhead
  - Memory
  - Time in context switches
  - Managing threads

- Programming issues
  - Communication between threads
  - Deadlock
  - Livelock
  - Multiple threads accessing same data

# Node.js

To be highly scalable it does not use:

- Threads*
- Blocking I/O

Instead uses callbacks

- When OS has data for you to read you callback function is called





*Well it does have processes and will use WebWorkers.

# Vertx.io

WebServer in Vertx.io

```
import org.vertx.java.core.Handler;
import org.vertx.java.core.http.HttpServerRequest;
import org.vertx.java.deploy.Verticle;

public class Server extends Verticle {
    public void start() {
        vertx.createHttpServer().requestHandler(new Handler<HttpServerRequest>() {
            public void handle(HttpServerRequest req) {
                String file = req.path.equals("/") ? "index.html" : req.path;
                req.response.sendFile("webroot/" + file);
            }
        }).listen(8080);
    }
}
```